Numerical Interpolation, Differentiation, and Integration

Read Chapter 1, Moin.

Main Idea:

- Approximate $f(x)$ by an expansion in terms of known functions, $\phi_j(x)$:

$$p(x) := \sum_{j=0}^{n} \hat{f}_j \phi_j(x) \approx f(x).$$

$\phi_j(x)$ – basis functions

$\hat{f}_j$ – basis coefficients.

- Often, we’ll denote the vector of basis coefficients as $\mathbf{\hat{f}} = [\hat{f}_0 \hat{f}_1 \ldots \hat{f}_n]^T$. 
• With this approximation, we also have an approximation to the derivative

\[ p'(x) \equiv \sum_{j=0}^{n} \hat{f}_j \phi_j'(x) = \sum_{j=0}^{n} \phi_j'(x)\hat{f}_j \approx f'(x). \]

• Typically, we are interested in the derivative at a fixed set of points, \( \tilde{x}_i, i = 0, \ldots, m. \)

• In this case, we have

\[ p'(\tilde{x}_i) = \sum_{j=0}^{n} \phi_j'(\tilde{x}_i)\hat{f}_j = \sum_{j=0}^{n} D_{ij}\hat{f}_j, \]

with the derivative matrix \( D \) having entries

\[ D_{ij} := \phi_j'(\tilde{x}_i). \]

• Notice that \( D \) does not depend on \( f(x) \).
• We can similarly approximate integrals via

\[
\int_a^b p(x) \, dx \equiv \int_a^b \sum_{j=0}^n \hat{f}_j \phi_j(x) \, dx
\]

\[
= \sum_{j=0}^n \left( \int_a^b \phi_j(x) \, dx \right) \hat{f}_j
\]

\[
= \sum_{j=0}^n w_j \hat{f}_j \approx \int_a^b f(x) \, dx.
\]

• Here, we have introduced the quadrature weights,

\[
w_j \ := \ \int_a^b \phi_j(x) \, dx.
\]
Basis Functions: $\phi_j(x), j = 0, \ldots, n.$

Examples:

- $\phi_j = x^j$ – Very poor choice and never used for $n > 3$.

- $\phi_j = e^{ij\alpha x}$ – Good for periodic functions on $[a, b]$ with 
  
  $\alpha = 2\pi/L, \ L = b - a, \ i = \sqrt{-1}.$

- $\{\phi_j\} = \{\cos 0x, \cos \alpha x, \sin \alpha x, \cos 2\alpha x, \sin 2\alpha x, \ldots\}$ 
  
  – Same as above but with real coefficients.

- $\phi_j = l_j$. \textit{Lagrange polynomial} (Lagrange cardinal function.)

  
  $$l_j(x) \in \mathbb{P}_n(x), \quad j = 0, \ldots, n$$

  
  $$l_j(x_i) = \delta_{ij} = \begin{cases} 
  1, & i = j \\
  0, & i \neq j 
  \end{cases}$$

  
  With this choice, the \textit{basis coefficients} are also \textit{nodal values}: $\hat{f}_j = f(x_j).$
• Piecewise polynomial Lagrange interpolants.

\[ \text{Lagrange interpolant} \iff \hat{f}_j = f(x_j). \]

• Splines (e.g., cubic spline).

• By far the most used are piecewise Lagrange polynomials.

• We will start with Lagrange polynomials spanning the interval \([a, b]\).

Figure 2: Examples of one-dimensional piecewise linear (left) and piecewise quadratic (right) Lagrangian basis functions, \(\phi_2(x)\) and \(\phi_3(x)\), with associated element support, \(\Omega^e, e = 1, \ldots, E\).
• For computation, most often use Lagrange interpolation:

\[ p(x) = \sum_{j=0}^{n} f_j l_j(x) \]

\[ l_j(x) \in \mathbb{P}_n \]

\[ l_j(x_i) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \]

• We construct the Lagrange (cardinal) basis functions as follows:

\[ l_i(x) = \alpha_i (x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n), \]

which clearly is in \( \mathbb{P}_n \) and satisfies \( l_i(x_j) = 0 \) when \( i \neq j \).

• To get the scaling condition, \( l_i(x_i) = 1 \), set

\[ \alpha_i = \left[(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)\right]^{-1}. \]

• A compact form for \( l_i(x) \) is

\[ l_i(x) = \prod_{j \neq i} \left( \frac{x - x_j}{x_i - x_j} \right) \]
• However, the first form involving $\alpha_i$ is generally faster for multiple queries (multiple evaluations with different $x$ values).

• To implement the fast form for a given $x$, first define

\[
\begin{align*}
s_0 &= 1 & t_n &= 1 \\
s_1 &= s_0 \cdot (x - x_0) & t_{n-1} &= t_n \cdot (x - x_n) \\
s_2 &= s_1 \cdot (x - x_1) & t_{n-2} &= t_{n-1} \cdot (x - x_{n-1}) \\
\vdots & & \vdots \\
s_n &= s_{n-1} \cdot (x - x_{n-1}) & t_0 &= t_1 \cdot (x - x_1)
\end{align*}
\]

then set $l_j(x) = \alpha_j \cdot s_j \cdot t_j$, for $j = 1, \ldots, n$.

• Notice that this form requires $O(n)$ operations for $n + 1$ outputs, i.e., the cost is linear in $n$.

• To be precise, the cost to interpolate from $n$ input points to $m$ output values is $O(n^2)$ for generating the full set of $\alpha_j$s, and $O(mn)$ for generating $m$ results.

• For certain choices of interpolation nodes (e.g., Chebyshev points), barycentric formulas can eliminate the $O(n^2)$ cost.

(See L.N. Trefethen, *Approximation Theory and Approximation Practice*, Oxford.)
Approximation Properties.
(Independent of method of construction.)

• $p$ is unique:
  Suppose $q(x) = f_j$ for $i = 0, \ldots, n$
  $q \in \mathbb{P}_n$

  \[
  d_n := p(x) - q(x) \in \mathbb{P}_n
  \]
  \[
  d_n(x_j) = 0 \quad j = 0, \ldots, n
  \]
  \[
  \Rightarrow n + 1 \text{ zeros}
  \]
  \[
  \Rightarrow d_n \equiv 0
  \]

• If $f$ has $m + 1$ continuous derivatives ($f \in C^{m+1}$) on $[a, b]$ with $[x, x_0, \ldots, x_n] \in [a, b]$, $\exists \xi \in (a, b)$ such that

  \[
  f(x) - p(x) = \frac{f^{n+1}(\xi)}{(n + 1)!} q_{n+1}(x)
  \]
  \[
  q_{n+1}(x) := (x - x_0)(x - x_1) \cdots (x - x_n)
  \]

  This is a most important error formula.
• If $f \in \mathbb{P}_n$ \quad $f^{n+1} \equiv 0 \quad \Rightarrow f \equiv p(x)$

• If $f = \alpha x^{n+1} + r_n(x)$ \quad $r_n \in \mathbb{P}_n$, then

$$f - p(x) = \frac{\alpha(n+1)!}{(n+1)!} q_{n+1}(x) = \alpha q_{n+1}(x).$$

$q_{n+1}(x)$ depends only on the nodal points \{x_j\} and not on $f$.

• In general, $q(x)$ might not be small.

• Also, $f^{n+1}(\xi)$ might not be small.

• Higher order does not imply smaller error.
  Careful choice of $x_j$’s, however, can imply $\max |q| \ll 1$.
  \(\Rightarrow\) Chebyshev points.
• Example: $n = 1$ – linear interpolation
• Huge advantage. We now have a continuous function,

\[ p(x) \simeq f(x) \]

parameterized by two numbers, and we know something about its error.

• If \( h := |x_1 - x_0| \) and \( x \in [x_0, x_1] \)

\[
M := \max_{\xi \in [x_0, x_1]} f''(\xi)
\]

\[
|f - p(x)| \leq \frac{m}{2} \max (x - x_0)(x - x_1)
\leq \frac{mL^2}{8}
\]

• Linear interpolation error is quadratic in grid spacing, \( h \).

• NOTE UNITS.
Some questions.

• In the finite element method (FEM), the Lagrangian interpolants are often called *shape functions*.

• We might ask what are the shapes of these functions?

• We will run the matlab demo code, `lagrange_bases.m` in a moment.

• First, let’s look at some examples.
Lagrange Basis Functions, $n=1$ (linear)

\[ l_0(x) \]

\[ l_1(x) \]
Lagrange Basis Functions, \( n=2 \) (quadratic)
Lagrange Polynomials: Good and Bad Point Distributions

$N=4$

$N=7$

$N=8$

Uniform

Gauss-Lobatto-Legendre
Unstable and Stable Interpolating Basis Sets

• Key idea for Chebyshev interpolation is to choose points that minimize \( \max |q_{n+1}(x)| \) on interval \( \mathcal{I} := [-1, 1] \).

\[
q_{n+1}(x) := (x - x_0)(x - x_1) \ldots (x - x_n)
\]

\[
:= x^n + c_{n-1}x^{n-1} + \ldots + c_0
\]

which is a monic polynomial of degree \( n + 1 \).

• The roots of the Chebyshev polynomial \( T_{n+1}(x) \) yield such a set of points by clustering near the endpoints.
• If \([a, b] = [-1, 1]\), Chebyshev points will minimize

\[ m \equiv \max_{x \in [-1, 1]} |q_{n+1}(x)|. \]

• Nodal points are zeros of \(T_{n+1}(x)\):

\[ x_i = -\cos \theta_i \quad \theta_i = \frac{\pi}{n} \left( i + \frac{1}{2} \right) \]

• There are many equivalent formulas for \(T_n(x)\):

\[ T_n(x) = \cos(n\theta) \quad \theta = \cos^{-1} x \]

\[ T_n(x) = \frac{1}{2} \left[ (x - \sqrt{x^2 - 1})^n + (x + (x^2 - 1))^n \right] \]

• Another is the three-term recurrence,

\[ T_0(x) = 1 \]
\[ T_1(x) = x \]
\[ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \]
**Nth-order Gauss-Chebyshev Points**

- Roots of Nth-order Chebyshev polynomial are projections of equispaced points on the circle, starting with $\theta = \delta \theta / 2$, then $\theta = 3\delta \theta / 2, ..., \pi - \delta \theta / 2$.

```matlab
% Gauss Lobatto Chebyshev Point generator
ti=(0:N)/(N); ti=pi*ti';
% Gauss Chebyshev Point generator
ti=(0:N); ti=(ti+.5)/(N+1); ti=pi*ti';
xi=cos(ti); yi=sin(ti);

close all; figure('Color',[1.0 1.0 1.0]);
for i=1:N+1;
    plot([xi(i) xi(i)], [0 yi(i)], 'ko', 'LineWidth', 2); hold on;
    plot([0 xi(i)], [0 yi(i)], 'ko-');
end;

N=100; % Draw Circle and x-axis:
ti=(0:N)/(N); ti=pi*ti'; % theta in [0, pi]
xi=cos(ti); yi=sin(ti); plot(xi,0*xi,'k-',xi,yi,'k-');
```
**N+1 Gauss-Lobatto Chebyshev Points**

N+1 GLC points are projections of equispaced points on the circle, starting with $\theta = 0$, then $\theta = \pi/N$, $2\pi/N$, ..., $k\pi/N$, ..., $\pi$.

\[ \Delta x_{\text{max}} \sim \frac{\pi}{N} \quad \Delta x_{\text{min}} \sim \frac{\pi^2}{2N^2} \]

$\delta \theta = \pi/N$
**Nth-Order Gauss Chebyshev Points**

Matlab Demo: `cheb_fun_demo.m`

```matlab
t=0:.01:(2*pi); t=t'; x=cos(t); y=sin(t);
n=9; z=cos(n*t);
plot3(x,y,z,'r','LineWidth',5); axis equal
```

\[
T_N(x) = \cos(N\theta)
\]

\[
x = \cos(\theta)
\]

\[
\cos(N\theta)
\]

\[
T_N(x)
\]
Here, we see the max $q_{n+1}$ for uniform (red) and Chebyshev points.

Chebyshev converges much more rapidly.
• matlab demo code, lagrange_bases.m

```matlab
clear all; close all; format compact
for n=2:2:100; no=10*n + 30;

% [z ,w ]=zwglc(n);  %% Gauss-Lobatto-Chebyshev points
% [zt,wt]=zwglc(no);

% [z ,w ]=zwgll(n);  %% Gauss-Lobatto-Legendre points
% [zt,wt]=zwgll(no);

[z ,w ]=zwuni(n);  %% Uniform points
[zt,wt]=zwuni(no);

J = interp_mat(zt,z);

j = floor(.2+n/2) + 1;  % "+1" because of matlab 1-based index
jn = 0*z; jn(j)=1;

xb= [0 0]; yb=[0 1];
plot(xb,yb,'k.-',zt,0*zt,'k-',zt,J(:,j),'r-',z,jn,'ro','linewidth',1.2);
drawnow;
pause

end;
```
Matlab Demos
Example Interpolation Applications

Differential Equations

• Consider the two-point boundary value problem (BVP):

\[-\tilde{u}''(x) = \sin \pi x, \quad x \in [0, 1]\]

\[\tilde{u}(0) = \tilde{u}(1) = 0.\]

• Exact solution is \(\tilde{u}(x) = \frac{1}{\pi^2} \sin \pi x.\)

• Approximate numerical solution:

\[u(x) = \sum_{j=0}^{n} l_j(x) u_j,\]

with \(u_0 = u_n = 0.\) (\(n - 1\) unknowns.)
• First derivative:

\[ u'(x) = \sum_{j=0}^{n} l'_j(x) u_j \iff u' = Du. \]

• Second derivative:

\[ u''(x) = \sum_{j=0}^{n} l'_j(x) u'_j \iff u'' = Du' = D^2u. \]

• **Collocation Method.** Apply differential equation at interior nodal points, \( x_j, j = 1, \ldots, n - 1 \).

• Set \( A \) to be the interior \((n-1) \times (n-1)\) components of the \((n+1) \times (n+1)\) matrix, \(-D^2\).

• In matlab: \( A=-D*D; \ A=A(2:end-1,2:end-1) \).
Matlab Code:

```matlab
clear all; close all; format compact

for n=2:20; nt=10*n + 20;
    [zn,wn]=zwgl1l(n);

    a= 0; b=1;
    x = a + (b-a)*0.5*(zn+1);

    D = deriv_mat(x);

    A = -D*D;
    A = A(2:end-1,2:end-1);

    f = sin(pi*x);
    f = f(2:end-1);

    u = A; u=[0; u ; 0];

    ut = sin(pi*x)/(pi*pi);

    en(n) = max(abs(u-ut));
    nn(n) = n;

    plot(x,ut,'ko-',x,u,'ro-','linewidth',1.2); pause(.1);
end;

figure
semilogy(nn,en,'ro','linewidth',2)
title('Error for -u_{xx} = sin \pi x','fontsize',16)
```
**Another Example**

- Consider 1D advection-diffusion with $c = 1$ and $q''' = 1$:

$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = \alpha \frac{\partial^2 u}{\partial x^2} + q'''$$

$$u(0) = 0, \quad u(1) = 0.$$ 

- Assume steady-state conditions $u_t = 0$

$$-\alpha u_{xx} + cu_x = 1, \quad u(0) = u(1) = 0.$$ 

- If $\alpha = 0$, we have:

$$cu_x = 1, \quad u(0) = u(1) = 0???$$

Too many boundary conditions!
The issue is that $\alpha \to 0$ is a *singular perturbation*.

This is true whenever the *highest-order derivative* is multiplied by a small constant.

As the constant goes to zero, the number of boundary conditions changes.

Here,

- We go from one boundary condition when $\alpha = 0$,
- to two boundary conditions when $\alpha > 0$ (even for $\alpha \ll 1$).

An example that is *not* a singular perturbation is

$$-u_{xx} + \epsilon u_x = 1, \quad u(0) = u(1) = 0.$$ 

This is called a *regular perturbation*. 
Matlab Demo: BVP-2

- Exact solution for our 1D model problem:

\[ u = \frac{x}{c} - \frac{L}{c} \left[ \frac{e^{cx/\alpha} - 1}{e^{cL/\alpha} - 1} \right] \]

\[ = \frac{1}{c} \left[ x - L \frac{e^{c(x-L)/\alpha} - e^{-cL/\alpha}}{1 - e^{-cL/\alpha}} \right]. \]

Let's modify our matlab code to solve this problem.
Next Step
Multidimensional Interpolation

- There are many strategies for interpolating $f(x,y)$ [ or $f(x,y,z)$, etc.].
- One easy one is to use tensor products of one-dimensional interpolants, such as bicubic splines or tensor-product Lagrange polynomials.

$$p_n(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{n} l_i(s) l_j(t) f_{ij}$$

2D Example: $n=2$